



New Signal Processing Libraries for Faust

Romain Michon, Julius Smith, Yann Orlarey

► **To cite this version:**

Romain Michon, Julius Smith, Yann Orlarey. New Signal Processing Libraries for Faust. Linux Audio Conference, 2017, Saint-Etienne, France. pp.83-87. hal-02159008

HAL Id: hal-02159008

<https://hal.archives-ouvertes.fr/hal-02159008>

Submitted on 18 Jun 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

New Signal Processing Libraries for Faust

Romain Michon, Julius Smith
CCRMA
Stanford University
Stanford, CA 94305-8180
USA
{rmichon,jos}@ccrma.stanford.edu

Yann Orlarey
GRAME
Centre National de Création Musicale
11 Cours de Verdun (Gensoul)
69002, Lyon
France
orlarey@grame.fr

Abstract

We present a completely re-organized set of signal processing libraries for the FAUST programming language. They aim at providing a clearer classification of the different FAUST DSP functions, as well as better documentation. After giving an overview of this new system, we provide technical details about its implementation. Finally, we evaluate it and give ideas for future directions.

Keywords

FAUST, Digital Signal Processing, Computer Music Programming Language

1 Introduction

FAUST is a functional programming language for real time Digital Signal Processing (DSP) targeting high-performance audio applications and plug-ins for a wide range of platforms and standards. [Orlarey et al., 2009]

One of FAUST’s strength lies in its DSP libraries implementing a large collection of reference implementations ranging from filters to audio effects and sound generators, etc.

When FAUST was created, it had a limited number of DSP libraries that were organized in a “somewhat” coherent way: `math.lib` contained mathematical functions, and `music.lib` everything else (filters, effects, generators, etc.). Later, the libraries `filter.lib`, `oscillator.lib`, and `effect.lib` were developed [Smith, 2008], [Smith, 2012], which had significant overlap in scope with `music.lib`.

A year ago, we decided to fully reorganize the FAUST libraries to

- provide more clarity,
- organize functions by category,
- standardize function names,
- create a dynamic documentation of their content.

In this paper, we give an overview of the organization of the new FAUST libraries, as well as technical details about their implementation. We then evaluate them through the results of a workshop on FAUST that was taught at the Center for Computer Research in Music and Acoustics (CCRMA) at Stanford University in 2016, and we provide ideas for future directions.

2 Global Organization and Standards

2.1 Overview

The new FAUST libraries¹ are organized in different files presented in Figure 1. Each file contains several subcategories allowing to easily find functions for specific uses. While some libraries host fewer functions than others, they were created to be easily updated with new elements. The content of the old (and now deprecated) FAUST libraries was spread across these new files, making backward compatibility a bit hard to implement (see §2.4).

More specifically, the old `music.lib` was removed since it contained much overlap in scope with `oscillator.lib`, `effect.lib`, and `filter.lib`.

`effect.lib` was divided into several “specialized” libraries: `compressors.lib`, `misceffects.lib`, `phaflangers.lib`, `reverbs.lib`, and `vaeffects.lib`. Similarly, the content of `oscillator.lib` is now spread between `noises.lib` and `oscillators.lib`. Finally, `demo.lib` hosts demo functions, typically adding user-interface elements with illustrative parameter defaults.

2.2 Prefixes

Each FAUST library has a recommended two-letter namespace prefix defined in the “meta library” `stdfaust.lib`. For example, `stdfaust.lib` contains the lines

¹<http://faust.grame.fr/library.html>. All the URLs in this paper were verified on 01/30/17.

<p>analyzer.lib</p> <ul style="list-style-type: none"> - Amplitude Tracking - Spectrum-Analyzers - Mth-Octave Spectral Level - Arbitrary-Crossover Filter - Banks and Spectrum Analyzers <p>basics.lib</p> <ul style="list-style-type: none"> - Conversion Tools - Counters and Time/Tempo Tools - Array Processing and Pattern Matching - Selectors (Conditions) - Other Misc Functions <p>compressors.lib</p> <p>Compressors and limiters library.</p> <p>delays.lib</p> <ul style="list-style-type: none"> - Basic Delay Functions - Lagrange Interpolation - Thiran Allpass Interpolation <p>demos.lib</p> <ul style="list-style-type: none"> - Analyzers - Filters - Effects - Generators <p>envelopes.lib</p> <p>Envelope generators library.</p> <p>filters.lib</p> <ul style="list-style-type: none"> - Basic Filters - Comb Filters - Direct-Form Sections - Direct-Form Second-Order - Biquad Sections - Ladder/Lattice - Virtual Analog Filters - Simple Resonator - Butterworth Filters - Elliptic (Cauer) Filters - Filters for Parametric Equalizers (Shelf, Peaking) - Arbitrary-Crossover Filter-Banks 	<p>maths.lib</p> <ul style="list-style-type: none"> - Constants - Functions <p>misceffects.lib</p> <ul style="list-style-type: none"> - Dynamic - Filtering - Time Based - Pitch Shifting - Meshes <p>noises.lib</p> <p>Noise generators library.</p> <p>oscillators.lib</p> <ul style="list-style-type: none"> - Wave-Table-Based Oscillators - LFOs - Low Frequency Sawtooths - Bandlimited Sawtooth - Bandlimited Pulse, Square, and Impulse Trains - Filter-Based Oscillators - Waveguide-Resonators <p>phaflangers.lib</p> <p>Phasers and flangers library</p> <p>reverbs.lib</p> <p>Reverbs library.</p> <p>routes.lib</p> <p>Signal routing library.</p> <p>signals.lib</p> <p>Misc signal tools library.</p> <p>spats.lib</p> <p>Spatialization tools library.</p> <p>synths.lib</p> <p>Misc synthesizers library.</p> <p>vaeffects.lib</p> <p>Virtual analog effects library.</p>
--	---

Figure 1: Overview of the organization of the new FAUST libraries.

```
fi = library("filters.lib");
os = library("oscillators.lib");
```

so that functions from `oscillator.lib` can be invoked using the `os` prefix and functions from `filter.lib` through `fi`:

```
import("stdfaust.lib");
process = os.sawtooth(440) : fi.lowpass
(2, 2000);
```

It is of course possible to avoid prefixes using the `import` directive:

```
import("filters.lib");
import("oscillators.lib");
process = sawtooth(440) : lowpass
    (2,2000);
```

The libraries presently avoid name collisions, so it is possible to load all functions from all libraries into one giant namespace soup:

```
import("all.lib");
process = sawtooth(440) : lowpass
    (2,2000);
```

Alternatively, all FAUST-defined functions can be loaded into a single namespace separate from the user's namespace:

```
sf = library("all.lib"); // standard
    faust namespace
process = sf.sawtooth(440) : sf.lowpass
    (2,2000);
```

Further details can be found in the documentation for the libraries.²

2.3 Standard Functions

The FAUST libraries implement dozens of functions, and it can be hard for new users to find standard elements for basic uses. For example, `filter.lib` contains seven different lowpass filters, and it's probably not obvious to someone with little experience in signal processing which one should be used.

To address this problem, the new FAUST libraries declare "standard" functions (see Figure 2) that are automatically added to the library documentation.³ Standard functions are organized by categories, independently from the library where they are declared (see §3). They should cover the needs of most users used to computer music programming environments such as PureData,⁴ SuperCollider,⁵ etc.

2.4 Backward Compatibility

With such major changes, providing a decent level of backward compatibility proved to be quite complicated. The old FAUST libraries (`effect.lib`, `filter.lib`, `math.lib`, `music.lib` and `oscillator.lib`) can still be used and will remain accessible for about one year.

In order to make this possible, we had to find a way to make them cohabit with the new libraries without creating conflicts. Thus, we decided to use plurals for the name of the new

²<http://faust.grame.fr/library.html>

³<http://faust.grame.fr/library.html>
#standard-functions.

⁴<https://puredata.info>.

⁵<http://supercollider.github.io>.

libraries, allowing to concurrently use our new `filters.lib` with the old `filter.lib`, for example.

If one of the old libraries is imported in a FAUST program, the FAUST compiler now throws a warning indicating the use of a deprecated library.

2.5 Other "Non-Standard" Libraries

A few "non-standard" libraries for very specific applications remain accessible but are not documented (see §3):

- `hoa.lib`: high order ambisonics library
- `instruments.lib`: library used by the FAUST-STK [Michon and Smith, 2011]
- `maxmsp.lib`: compatibility library for Max/MSP
- `tonestacks.lib`: tonestack emulation library used by Guitarix⁶
- `tubes.lib`: guitar tube emulation library used by Guitarix

3 Automatic Documentation

The new FAUST libraries use a new automatic documentation system based on the `faust2md` (FAUST to Markdown) script which is now part of the FAUST distribution. It allows to easily write Markdown comments within the code of the libraries by respecting the standards described below.

Library headers and descriptions can be created with

```
##### Library Name ##### // Some
    Markdown text.
#####
```

Libraries can be organized into sections using the following syntax:

```
===== Section Name ===== // Some
    Markdown text.
=====
```

Each function in a library should be documented as such:

```
---- Function Name ---- // Some
    Markdown text.
-----
```

The libraries documentation can be conveniently generated by running:

```
make doclib
```

⁶<http://guitarix.org>.

Analysis Tools		Envelopes	
an.amp_follower	Amplitude follower	en.adsr	ADSR envelope
an.mth_oct [...]	Octave analyzers	en.ar	AR envelope
		en.asr	ASR envelope
		en.smoothEnv	Exponential envelope
Basic Elements		Filters	
ba.beat	Pulse generator	fi.bandpass	Bandpass (Butterworth)
si.block	Block a signal	fi.resonbp	Bandpass (resonant)
ba.bpf	Break Point Function	fi.bandstop	Bandstop (Butterworth)
si.bus	Bus of n signals	fi.tf2	Biquad Filters
ba.bypass1	Bypass (mono)	fi.allpass_fcomb	Comb (allpass)
ba.bypass2	Bypass (stereo)	fi.fb_fcomb	Comb (feedback)
ba.count	Counts in a list	fi.ff_fcomb	Comb (feedforward)
ba.countdown	Samples count down	fi.dcblocker	DC blocker
ba.countup	Samples count up	fi.filterbank	Filterbank
de.delay	Integer delay	fi.fir	FIR (arbitrary order)
de.fdelay	Fractional delay	fi.high_shelf	High shelf
ba.impulsify	Signal to impulse	fi.highpass	Highpass (Butterworth)
ba.sAndH	Sample and hold	fi.resonhp	Highpass (resonant)
ro.cross	Cross n signals	fi.iir	IIR (arbitrary order)
si.smoo	Smoothing	fi.levelfilter	Level filter
si.smooth	Controllable smoothing	fi.low_shelf	Low shelf
ba.take	Element from a list	fi.lowpass	Lowpass (Butterworth)
ba.time	Timer	fi.resonlp	Lowpass (resonant)
		fi.notchw	Notch filter
		fi.peak_eq	Peak equalizer
Conversion		Generators	
ba.db2linear	dB to linear	os.impulse	Impulse
ba.linear2db	Linear to dB	os.imptrain	Impulse train
ba.midikey2hz	MIDI key to Hz	os.phasor	Phasor
ba.pole2tau	Pole to t60	no.pink_noise	Pink noise
ba.samp2sec	Samples to seconds	os.pulsetrain	Pulse train
ba.sec2samp	Seconds to samples	os.lf_imptrain	Low-freq pulse train
ba.tau2pole	t60 to pole	os.sawtooth	Sawtooth wave
		os.lf_saw	Low-freq sawtooth
		os.osc	Sine (filter-based)
		os.oscsin	Sine (table-based)
		os.square	square wave
		os.lf_square	Low-freq square
		os.triangle	Triangle
		os.lf_triangle	Low-freq triangle
		no.noise	White noise
Effects		Synths	
ve.autowah	Auto-wah	sy.additiveDrum	Additive drum
co.compressor	Compressor	sy.dubDub	Filtered sawtooth
ef.cubicnl	Distortion	sy.combString	Comb string
ve.crybaby	Crybaby	sy.fm	FM
ef.echo	Echo	sy.sawTrombone	Lowpassed sawtooth
pf.flanger	Flanger	sy.popFiltPerc	Popping filter
ef.gate_mono	Signal gate		
co.limiter	Limiter		
pf.phaser2	Phaser		
re.fdnrev0	Reverb (FDN)		
re.freeverb	Reverb (Freeverb)		
re.jcrev	Reverb (simple)		
re.zita_rev1	Reverb (Zita)		
sp.panner	Panner		
ef.transpose	Pitch shift		
sp.spat	Panner		
ef.speakerbp	Speaker simulator		
ef.stereo_width	Stereo width		
ve.vocoder	Vocoder		
ve.wah4	Wah		

Figure 2: Standard FAUST functions with their corresponding prefix when used with `stdfaust.lib`.

at the root of the FAUST distribution. This will generate an html and a pdf file in the /documentation folder using pandoc.⁷

4 Evaluation and Future Directions

The new FAUST libraries were beta tested during the *CCRMA Faust Summer Workshop* at Stanford University.⁸ In previous editions of the workshop, students had to go through the library files to get the documentation of specific functions. During last year's workshop, thanks to the new libraries documentation, students were able to find information about functions simply by doing a search in the documentation file. Additionally, none of them encountered problems while using the new libraries which was very satisfying.

The FAUST libraries are meant to grow with time, and we hope that this new format will facilitate the integration of new contributions. Eventually, we plan to divide `filters.lib` into more subcategories, like we did for the old `oscillator.lib`. Finally, `physmodels.lib` which is a new library for physical modeling of musical instruments is currently under development.

5 Conclusions

The new FAUST libraries provide a platform to easily prototype DSP algorithms using the FAUST programming language. Their new organization, in combination with their automatically generated documentation, simplifies the search for specific elements covering a wide range of uses. New “standard functions” help to point new users to useful elements to implement various kind of synthesizers, audio effects, etc. Finally, we hope that this new format will encourage new contributions.

6 Acknowledgments

Thanks to Albert Gräf for his contributions to the design of the new libraries, and for single-handedly implementing a solid backward-compatibility scheme!

References

Romain Michon and Julius O. Smith. 2011. Faust-STK: a set of linear and nonlinear physical models for the Faust programming

language. In *Proceedings of the 14th International Conference on Digital Audio Effects (DAFx-11)*, Paris, France, September.

Yann Orlarey, Stéphane Letz, and Dominique Fober, 2009. *New Computational Paradigms for Computer Music*, chapter “Faust : an Efficient Functional Approach to DSP Programming”. Delatour, Paris, France.

Julius Orion Smith. 2008. Virtual electric guitars and effects using Faust and Octave. In *Proceedings of the Linux Audio Conference (LAC-08)*, pages 123–127, KHM, Cologne, Germany.

Julius O. Smith. 2012. Signal processing libraries for Faust. In *Proceedings of Linux Audio Conference (LAC-12)*, Stanford, USA, May.

⁷<http://pandoc.org>.

⁸<https://ccrma.stanford.edu/~rmichon/faustWorkshops/2016>.